(12) UK Patent Application (19) GB (11) 2 244 356 (13) A

(43) Date of A publication 27.11.1991

(21) Application No 9107865.9

(22) Date of filing 12.04.1991

(30) Priority data
(31) 9008366 (32) 12.04.1990 (33) GB

(71) Applicant
British Aerospace Public Limited Company

(Incorporated in the United Kingdom)

11 Strand, London, WC2N 5JT, United Kingdom

(72) Inventors
Hugo R Simpson
Eric R Campbell

(74) Agent and/or Address for Service
E C Dowler
British Aerospace plc, Corporate IPR Dept,
PO Box 87, Royal Aerospace Establishment,
Farnborough, Hants, GU14 6YU, United Kingdom

(51) INT CL[5]
G06F 13/368

(52) UK CL (Edition K)
G4A AFGL

(56) Documents cited
GB 1490612 A EP 0123509 A2 WO 84/00221 A1

(58) Field of search
UK CL (Edition K) G4A AFGK AFGL AFGX
INT CL[5] G06F

(54) Inter-processor communication

(57) A multiple processor computer system in which a group of computing units, each with a processor 41 linked to a private memory 43 *via* a private data bus, are linked each one to every other *via* a respective separate independent shared memory area 46 controlled by a communications controller 48 which can make available a full asynchronous two way communication route through the memory area. Multi-tasking capability of the computer units is controlled by a further set of unit controllers 47 in combination with respective software task schedulers.

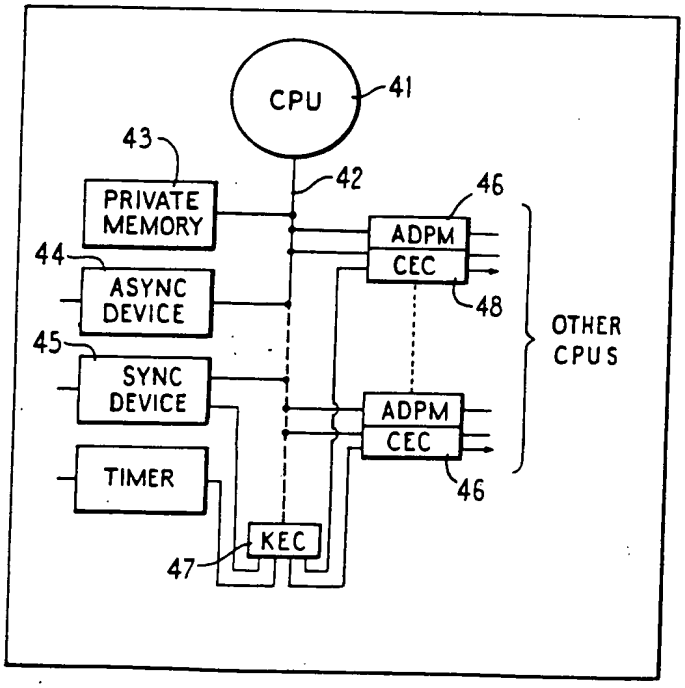

FIG.2

At least one drawing originally filed was informal and the print reproduced here is taken from a later filed formal copy.

GB 2 244 356 A


READER
SHARED MEMORY
WRITER

FIG.1

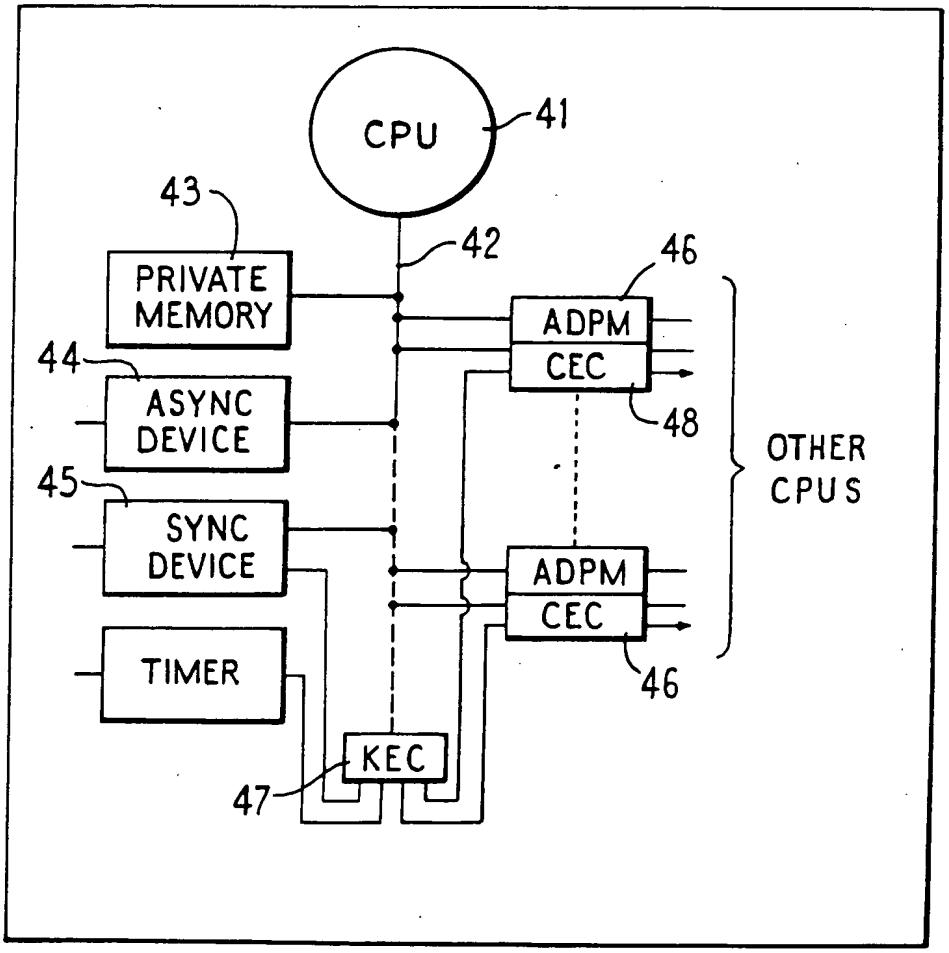
CPU
41
43
42
46
PRIVATE MEMORY
ADPM
CEC
44
ASYNC DEVICE
48
OTHER CPUS
45
SYNC DEVICE
ADPM
CEC
TIMER
46
KEC
47

FIG.2

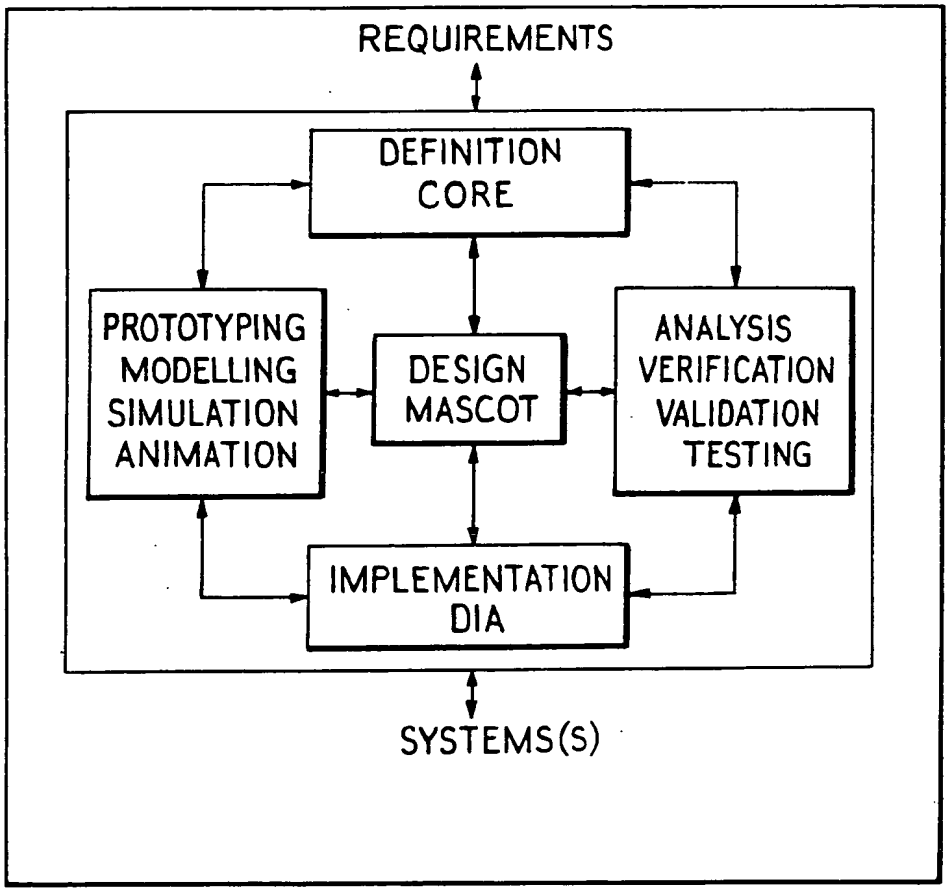
REQUIREMENTS
DEFINITION
CORE
PROTOTYPING
MODELLING
SIMULATION
ANIMATION
DESIGN
MASCOT
ANALYSIS
VERIFICATION
VALIDATION
TESTING
IMPLEMENTATION
DIA
SYSTEMS(S)

FIG.3

COMPUTERS

There is a continuing growth in the amount of computer power required to support digital data processing applications. One response to this problem is to develop larger, faster and more complex single processors; another is to couple multiple processors together, for example by high speed data buses.

With existing forms of computer systems using intercommunicating multiple processors, various problems arise and the object of the present invention is to provide an alternative multi-processor system, preferably incorporating an integrated associated hardware/software system concept, which may be preferred for some application areas. According to the invention, there is provided a distributed computer system comprising:-

a plurality of asynchronous computer units each with a data processor and a private data memory linked to the processor by way of a private data bus;

shared data memory means having access ports linked to respective ones of the computer units via the associated private data busses for each computer unit to be able to communicate with any other computer unit by way of a respective two-way data route comprising a respective individual area of shared data memory; and

communication control means connected to the private data busses and the shared data memory means for responding to control data issued by the processor of any computer unit to initiate communication via a route determined by the processor.

For a better understanding of the invention, reference will be made, by way of example, to the accompanying drawings, in which:-

figure 1 is a diagram for explaining the general nature of inter-process communication used in the present invention;

figure 2 is a simplified diagram of part of a computer system; and

figure 3 is a diagram for explaining the use of the figure 2 system in the context of an integrated hardware/software development environment.

In connection with the system to be described, reference will be made to a so-called Data Interaction Architecture (DIA), following the emphasis on the data which lies between concurrent system processes. The term Architecture is used in the sense of the elements, and their interconnection and grouping, which go to make up a digital data processing system. Essentially these elements are the software processes and hardware processors which communicate through shared data areas declared in shared memory. Thus, the DIA covers multi tasking software as well as multi processing hardware implementation. The DIA can be seen as an integrating technology which provides a framework for system and component design. It is a general approach which is usable with a wide range of processor types and programming languages.

In a computer system, inter-process and inter-processor communication may be direct, ie in total synchronism with the "reading and writing" processes being locked together at the point of communication. Data independent of the processes cannot exist since there is no data area (or process) which can hold information in transit. This is the rendezvous style of communication which naturally introduces severe timing interdependences between the two processes. A monitor process can be interposed in the communication path so as, in effect, to decouple the operation of

the reader and writer processes, but this is only at the expense of significant additional overheads and cannot entirely remove the timing interactions.

The system to be described, ie DIA, comprises a real time network where communication via shared memory, ie it is indirect as shown in Figure 1. This form of communication is more flexible than those referred to above in that it can be used to provide a wide range of communication protocols including fully and conditionally asynchronous, loosely synchronous (the bounded buffer) and fully synchronous (the rendezvous) forms. Asynchronous and loosely synchronous protocols avoid the tight interlocked timing relationships implicit in the rendezvous, and significantly reduce the risks of deadlock and severe performance degradation at run time. However DIA does not prejudge the optimum implementation form; all protocols are supported so that the system designer can select the most appropriate for the application in hand.

Software structure in DIA is modelled on the known MASCOT (Modular Approach to Software Construction Operation and Test) form of real time network. MASCOT is a software design method based on data flow concepts and is described, for example, in the articles "Process Synchronisation in Mascot" by H R Simpson and K Jackson, Computer Journal, 1979, 22, (4), pp 332-345 and in "The Mascot Method" by H R Simpson, Software Engineering Journal, 1986, 1, (3), pp 103-120. It has the important advantage of allowing the distribution of system functionally to be represented, so providing the means both of controlling the mapping of software designs into distributed hardware and of allowing real time

properties to be analysed in terms of information propagation effects.

Individual MASCOT processes are known as ACTIVITIEs. Each ACTIVITY is conceptually independent, ie it runs concurrently with all other ACTIVITIEs. In practice, where ACTIVITIEs share a processor, a scheduler must be provided together with the synchronisation primitives to support mutual exclusion and cross stimulation.

MASCOT shared data areas, through which the ACTIVITIEs communicate, are known as Intercommunication Data Areas (IDAs) and there are two principal classes. The POOL form of IDA is used to hold reference data which is maintained by one or more updating processes to be consulted by one or more using processes with minimal timing interference. The CHANNEL form of IDA is used to pass messages between one or more producing processes to one or more consuming processes. POOLs are essentially asynchronous whereas CHANNELs are synchronous.

An important DIA extension to MASCOT is the ROUTE concept. A ROUTE is used to provide communication between a single writing process and a single reading process, and it is equivalent to either a POOL (asynchronous communication between an updater and a user) or a CHANNEL (synchronous communication between a producer and a consumer). ROUTEs are used to express abstract communication designs and can be mapped into the hardware in a variety of forms which meet the communication requirements regardless of the relative location of the ACTIVITIEs connected by the ROUTE. DIA provides special executive software and hardware facilities to support the ROUTE concept.

The DIA processing configuration is shown in Figure 2. Ideally the Central Processing Unit (CPU) 41 is a relatively simple form of Reduced Instruction Set Computer (RISC) in which no use is made of features which introduce non deterministic timing effects such as interrupts, caching and the like. More complex computers can be used but this will make it more difficult to analyse run time properties.

The central vertical line 42 in Figure 2 depicts the CPU's private memory bus. This gives access to: Private Memory (containing private network elements) 43, Asynchronous Devices (peripherals) 44, Synchronous Devices (peripherals which can generate an external stimulus 45, a series of Asynchronous Dual Port Memories (ADPM - containing shared IDA elements) 46 and two sorts of specialised VLSI devices, namely a Kernel Executive Chip (KEC) 47 and for each memory 46, a Comms Executive Chip (CEC) 48.

The KEC supports the multi tasking facilities needed when many activities are mapped into a single processor (Processor = CPU + private memory). It is also able to accept external stimuli which are demands for processing arising outside the processor (eg from Synchronous Devices, Timers, CECs, etc). External stimuli can be regarded as cooperative interrupts; they allow external demands to be taken into consideration at each reschedule point (the success of this strategy is clearly dependent on the accurate prediction of maximum slice times, and on the provision of special processors to handle any fast reaction time requirements). Different functions on the KEC are invoked by write or read access to different addresses assigned to the chip (this allows the chip to provide its function when interfaced to a wide variety of CPU types).

Communication with an adjacent processor is provided by an ADPM-CEC pair. Each ADPM has two entirely independent access paths to the memory locations, thus avoiding the need for any form of arbitration at the basic hardware level (data integrity is maintained by the CEC and software executive (see below)). Like the KEC, selection of functions on the CEC is by means of write and read operations to specific addresses, with an additional facility to select one of many CECs by use of a unique value in the data field of a write operation. Some CEC operations generate external stimuli which are passed to the KEC of an adjacent processor (this is needed for the synchronous ROUTEs between adjacent processor pairs). The two interfaces of a CEC, one for each of the connected processors, provide identical facilities and, like the ADPM, no arbitration is needed.

Multi tasking facilities, in the shape of KEC and software scheduler, provide the means by which the CPU in each processor is shared between resident ACTIVITIEs.

The KEC is able to register demands for processing and to select the next ACTIVITY to be allocated CPU time. The KEC currently available provides scheduling support for up to 64 ACTIVITIEs arranged in 8 priority levels with 8 ACTIVITIEs at each level. External stimuli are routed through (indirectly) to the highest priority level. The selection strategy consists of choosing an ACTIVITY from the highest priority level containing a demand for processing; where there is more than one demand at this level then a round robin search is used to select the next ACTIVITY (the chip remembers the last ACTIVITY scheduled at each level). The chip indicates the next ACTIVITY to be scheduled by

returning a number in the range 0..63; if there is no current demand then 64 is returned.

The KEC contains two primary control bits for each ACTIVITY. The first is the start bit which must be set for an ACTIVITY to be candidate for scheduling. This provides an overall control and can be used to implement the MASCOT control commands. The second bit is the stim bit which is used to denote a current demand for scheduling. External stimuli are held on the chip and are entered into the schedule at a suitable point (see below).

The principal form of interaction between the KEC and the software scheduler make use of the following chip operations (all of which execute in a single memory access, although they are portrayed as PROCEDUREs or FUNCTIONs for the purposes of explanation):-

a. PROCEDURE kec_suspend; The external stimuli are accepted into the schedule (by setting the stim bit for any outstanding external demand). The stim bit for the current activity is reinstated, registering a request for further processing.

b. PROCEDURE kec_wait; As for kec_suspend except that the stim bit is not reinstated.

c. PROCEDURE kec_stim (act : 0..63); This supports internal stimuli whereby one ACTIVITY can set the stim bit of another.

d. FUNCTION kec_nextact : 0..64; This returns the number of the next ACTIVITY to be scheduled and it clears the external stimulus demands where these have been accepted into the schedule, and it then clears the stim bit of the ACTIVITY selected for scheduling.

The external stimuli are potentially asynchronous and the KEC and associated software ensure that the attendant metastability hazard is reduced to a negligible level (it is effectively eliminated). This is achieved by the delay which must exist between acceptance of the external stimuli by kec_suspend or kec_wait, and the use of kec_nextact to select the next ACTIVITY.

The software scheduler which interfaces with the KEC is particularly straightforward. First we introduce some auxiliary definitions:

a. VAR curract : 0..64; This is a variable which holds the number of the currently scheduled ACTIVITY.

b. PROCEDURE save; This procedure saves the context of the ACTIVITY whose number is indicated by the value of curract. It is assumed that space has been set aside for this purpose. The context of an ACTIVITY is initialised so that the ACTIVITY is first entered at its start point.

c. PROCEDURE restore; This procedure restores the context of the ACTIVITY whose number is indicated by the value of curract.

Scheduling primitives which interface directly with the KEC by the value can now be formulated:-

a.
```
PROCEDURE suspend;
BEGIN
  kec_suspend;
  save;
  curract := kec_nextact;
  restore
END;
```

b. PROCEDURE wait;

```
BEGIN
  kec_wait;
  save;
  curract := kec_nextact;
  restore
END;
```

c.
```
PROCEDURE stim (act : 0..63);
BEGIN
  kec_stim (act)
END;
```

We are now in a position to see how cross stimulation and mutual exclusion, the basic synchronisation primitives, can be provided.

A control_node record type is introduced to provide a control point at which an ACTIVITY may wait, to be stimmed into operation by another ACTIVITY:-

a.
```
TYPE control_node =
RECORD
  activity : 0..63;
  waiting : BOOLEAN
END;
```

where waiting is initialised to FALSE. The cross stimulation facility is provided thus:

b.
```
PROCEDURE wait_cn (VAR cn : control_node);
BEGIN
  cn.activity := curract;
  cn.waiting := TRUE;
  wait
END;
```

c. PROCEDURE stim_cn (VAR cn : control_node);

```
BEGIN
  IF cn.waiting THEN
  BEGIN
    stim (cn.activity);
    cn.waiting := FALSE
  END
END;
```

Mutual exclusion is a little more elaborate and some auxiliary definitions are needed:-

a. TYPE act_queue; This is the type of variable capable of holding a FIFO queue of ACTIVITY numbers.

b. PROCEDURE add_back (VAR q : act_queue); Adds the current ACTIVITY to the back of the designated act_queue.

c. FUNCTION take_front (VAR q : act_queue) : 0..63; Takes the ACTIVITY off the front of the designated act_queue.

A control_queue record type is introduced to provide points at which ACTIVITIEs may be held in a FIFO queue pending the availability of a 'resource' which is also needed by other ACTIVITIEs:-

a. TYPE control_queue =

```
RECORD
  count : INTEGER;
  queue : act_queue
END;
```

where count is initialised to -1 and the queue is initialised to empty. The mutual exclusion facility is provided thus:-

b. PROCEDURE join_cq (VAR cq : control_queue);

```
BEGIN
  cq.count := cq.count + 1;
  IF cq.count <> 0 THEN
  BEGIN
    add_back (cq.queue);
    wait
  END
END;
```

c.
```
PROCEDURES leave_cq (VAR cq : control_queue);
BEGIN
  IF cq.count <> 0 THEN
    stim (take_front (cq.queue));
  cq.count := cq.count - 1
END;
```

The cross stimulation and mutual exclusion primitives provide all that is needed to support synchronous interactions within a single processor. They are compact and simple to implement.

Communication facilities, in the shape of CEC, ADPM, ROUTE designs and software executive, provide the means by which ACTIVITIEs in adjacent processors pass data from one to another. Our description here will concentrate on this particular shared memory configuration but it must be stressed that a ROUTE is a design abstraction which, without change to interface or process interaction properties, can also represent communications between ACTIVITIEs within a single processor, and communication between ACTIVITIEs located in processors which have no shared memory. Also the ROUTE is but one possible form of IDA communication, and alternative designs will often be needed to meet particular application requirements.

The interfaces to a ROUTE may be either procedural or data, and in each case single items are inserted or extracted, and pass through the ROUTE unchanged, ie the operation of communicating through a ROUTE has no semantic effect whatsoever. However there are various dynamic possibilities:-

a. Fully Asynchronous. The ROUTE is effectively a POOL where the writing and reading ACTIVITIEs can insert or extract data at any time, and these operations can be of any duration. The communication protocol is that of the four slot mechanism (see EP Patent Specification No 0292287) and data coherence and freshness are guaranteed. This is known as a fs_route.

b. Conditionally Asynchronous. The ROUTE is effectively a POOL operating a swung buffer protocol. Data coherence is guaranteed provided that the duration of reads is always less then the interval between writes and vice versa. This is known as a ts_route.

c. Loosely Synchronous. The ROUTE is effectively a two item MASCOT standard CHANNEL. It provides a message passing facility with a limited amount of buffering. This is known as a bb_route.

d. Fully Synchronous. The ROUTE is effectively operated as a rendezvous between the communicating processes. It provides a message passing facility with no apparent buffering (although the ADPM space requirements for loosely and fully synchronous forms are identical). This is know as a rv_route.

As has already been mentioned, a given CEC is selected when its unique number appears in the data field of a particular write operation (at the same time all other CECs are deselected). In addition to the chip number in this write operation, a channel

number is also selected. Each side of the current chip contains 16 channels, with each side of each channel containing the logic for:-

a. counter (two bit) stepping

b. counter (two bit) comparison

c. transmit stim (multiplexed) X 2

d. receive stim (multiplexed) X 2

e. two slot async send

f. two slot async receive

g. four slot aysnc send

h. four slot async receive

This logic supports:-

a. 16 X bb-route OR rv_route, left to right OR right to left. The CEC logic allows for a total of 16 possible synchronous ROUTES (bb_route or rv_route), each of which either passes data in one direction or the other. The choice of ROUTE type and direction is exercised when the CEC logic is allocated to application communication functions. The counter stepping and inter processor stims for each channel are integrated on the chip to give the most compact operation (see below).

b. 16 X stim_only, left to right AND right to left. A further 16 inter processor stims in both directions are provided to allow additional synchronous ROUTEs to be built (by software).

c. 16 X ts_route and fs_route, left to right AND right to left. The CEC logic allows for 16 fully and 16 conditionally asynchronous ROUTEs in both directions, 64 ROUTEs in all.

The chip and channel selection operation, needed at the start of any communication procedure or data access involving the CEC, also sets the 'mode' of the chip so that it can execute the

appropriate individual operations in support of a particular communication protocol. The mode is defined as follows:-

a. TYPE mode = (stims, fs_rd, fs_wr, init, ts_wr, ts_rd, test, sync);

The selection operation can now be written:-

a. PROCEDURE cec_select (chip : 0..7; chan : 0..31; m : mode); The chip parameter lies in the range 0..7 because the current CEC allows 1 of 8 chips (all at the same address) to be selected. The chan parameter lies in the range 0..31 because transmit and receive stim facilities are 32 channels wide; 16 stims in each direction are intimately associated with the counter stepping logic, with a further 16 in each direction supporting stim_only.

All types of ROUTE require data space to be allocated within the ADPM associated with the CEC. The reasonably large number and variety of ROUTE types supported by the CEC allows considerable flexibility in the choice of ROUTEs. Like the KEC, CEC operations all execute in a single memory access.

The CEC provides a number of operations to support a four slot fully asynchronous protocol:-

a. PROCEDURE cec_fs_rd_prl; This is the operation which chooses a slot pair for reading.

b. PROCEDURE cec_fs_rd_pr2; This is the operation which chooses the slot within the pair for reading.

c. FUNCTION cec_fs_rd_slot : 0..3; This is the operation which returns the number of the slot to be read from next. It is dependent on the results from the previous two operations both of which are potentially asynchronous, and hence in theory metastability is a hazard. In practice, at computer rates of

operation, the chip design and the delay before the slot number is read are such that metastability is effectively eliminated.

d. PROCEDURE cec_fs_wr_pw1; This is the operation which indicates the slot containing the latest data in the chosen pair, and which also determines the slot in the chosen pair that will be written to next.

e. PROCEDURE cec_fs_wr_pw2; This is the operation which indicates the pair which contains the latest data and which chooses the pair that will be written to next.

f. FUNCTION cec_fs_wr_slot; This is the operation which returns the number of the slot to be written to next. It is dependent on the results from the previous two operations, the second of which is potentially asynchronous and hence again is theoretically vulnerable to metastability. In practice the chip design and method of use eliminate this hazard.

All these operate on the chip, channel and mode preselected by cec_select.

A fully asynchronous ROUTE requires an appropriate four slot array to be declared in the ADPM, and we will assume that the data to be passed is of type DATA. The ROUTE can be represented thus:-

a. VAR data : ARRAY[0..3] OF DATA;

```
FUNCTION fs_read : DATA;
BEGIN
  cec_select (chip, chan, fs_rd);
  cec_fs_rd_pr1;
  cec_fs_rd_pr2;
  fs_read := data [cec_fs_rd_slot]
```

```
END;
```

b.
```
PROCEDURE fs_write (VAR item : DATA);
BEGIN
  cec_select (chip, chan, fs_wr);
  data [cec_fs_wr_slot] := item;
  cec_fs_wr_pw1;
  cec_fs_wr_pw2
END;
```

These reading and writing operations illustrate the interactions with the CEC. They do not indicate the way in which the appropriate chip and chan parameters are associated with the cec_select calls; this is arranged by the network building software and is beyond the scope of this paper. A further important point concerns initialisation, and it is necessary to ensure that the data array in the ADPM is initialised so that any read occurring before the first write will not receive erroneous values.

The two slot conditionally asynchronous protocol has its own special operations. On the reading side there is one operation to choose the slot and one to return the number of the next slot to be read. On the writing side, the indication of the latest data and the return of the slot number for writing are combined into a single operation. This means that the writing procedure must remember the slot number between calls (indicated by the OWN variable below). The ROUTE can be represented thus (assuming appropriate initialisation and cec_select parameterisation):-

a.
```
VAR data : ARRAY [0..1] OF DATA;

FUNCTION ts_read : DATA;
```

```
BEGIN
  cec_select (chip, chan, ts_rd);
  cec_ts_rd_pr;
  ts_read := data [cec_ts_rd_slot]
END;

PROCEDURE ts_write (VAR item : DATA);
OWN next : 0..1;
BEGIN
  cec_select (chip, chan, ts_wr);
  data [next] := item;
  next := cec_ts_wr_pw_slot
END;
```

The mechanism for handling inter processor stims involves CEC and executive software functions. There are two levels of external stim, primary and secondary. Primary stims are generated by CEC operations on one side of the chip and are transmitted through to the other to be held on the KEC whence they are introduced into the schedule and can cause an ACTIVITY to run (see above). There are 32 secondary stims associated with each primary stim and an operation is provided to interrogate them:-

a. FUNCTION cec_next : 0..32; This is used to search for secondary stims. The set of outstanding secondary stims is latched on the preceding cec_select operation and each channel is examined in turn. When a stim is found its number is returned, and at the same time it is cleared. When there are no further secondary stims the number 32 is returned.

This operation is used by an executive ACTIVITY whose function it is to pass the secondary stim through to an appropriate

control_node where it will in turn cause an application ACTIVITY to be scheduled. This can be represented thus:-

a. VAR xstim : ARRAY [0..7, 0..31] OF control_node;

```
ACTIVITY exec;
VAR next : 0..32;
BEGIN
  WHILE TRUE DO
  BEGIN
    cec_select (chip, chan, stims);
    next := cec_next;
    WHILE next <> 32 DO
    BEGIN
      stim_cn (xstim [chip, next]);
      next := cec_next
    END;
    wait
  END
END;
```

An exec ACTIVITY must be installed for each CEC, and its initial context must be set into the context saving area so that it is scheduled as a result of the first appropriate primary external stim. Thereafter it will wait whenever it has completed the task of interrogating the secondary stims, and having rescheduled the relevant application ACTIVITIEs via the xstrim array. Clearly there will be some uncertainty as to the time taken between the raising of an external stim in one processor and its use to schedule an ACTIVITY in another. The outer bound of this delay is calculable from a knowledge of the longest slice time (ie interval

between reschedule points) of all ACTIVITIEs in a processor, together with the slice times of the other ACTIVITIEs at the highest priority level. The xstim declaration assumes a full complement of 8 CECs with a need for 32 channels on each. It is extremely unlikely that this capacity could ever be serviced by a single processor and the space allocated for these control_nodes would be kept to just that required for the application in hand.

The synchronous ROUTEs between adjacent processors make use of the inter processor stim facility just described, and in addition they are supported by CEC logic in the form of two bit counters, together with counter stepping and testing operations:-

a. FUNCTION cec_inc_stim; This increments the counter for this side (ie from which the operation is executed) of the selected channel on the selected chip, and it generates an external stim (both levels). A number in the range 0..1 is returned this being the new counter value MOD 2, indicating the slot to be next accessed for data transfer.

b. FUNCTION cec_sync_p0; This returns 0 if the counter on this side equals the counter on the other side plus zero, ie the two counters are the same; otherwise 1 is returned.

c. FUNCTION cec_sync_p2; This returns 0 if the counter on this side equals the counter on the other side plus two; otherwise 1 is returned.

The counters each step through the range 0..3 and are used to indicate full and empty conditions in a two slot MASCOT standard CHANNEL. Initialisation of the counters is effected using an operation which steps the counter without generating a stim. For a bounded buffer ROUTE the counters are initialised to zero, and the xstim array elements (control_nodes) associated with a synchronous

ROUTE must be initialised to the 'unstimmed' state. The ROUTE can be represented thus (assuming appropriate cec_select parameterisation and OWN variables initialised to zero):-

a. VAR data : ARRAY [0..1] OF DATA;

```
FUNCTION bb_read : DATA;
OWN oc : 0..1;
BEGIN
  cec_select (chip, chan, sync);
  WHILE cec-sync_p0 = 0 DO
  BEGIN
    wait_cn (xstim [chip, chan]);
    cec_select (chip, chan, sync)
  END;
  bb_read := data [oc];
  oc := cec_inc_stim
END;

PROCEDURE bb_write (VAR item : DATA);
OWN ic : 0..1;
BEGIN
  cec_select (chip, chan, sync);
  WHILE cec_sync_p2 = 0 DO
  BEGIN
    wait_cn (xstim [chip, chan]);
    cec_select (chip, chan, sync)
  END;
  data [ic] := item;
  ic := cec_inc_stim
```

```
END;
```

The implementation of a fully synchronous ROUTE is found to be very similar to a loosely synchronous ROUTE. The CEC must be initialised so that the counter on the writing side is 1 and the counter on the reading side is 0. Likewise the ic OWN variable must be initialised to 1. The ROUTE is represented thus:-

a.
```
VAR data : ARRAY [0..1] OF DATA;
FUNCTION rv_read : DATA;
VAR oc : 0..1;
BEGIN
  cec_select (chip, chan, sync);
  oc := cec_inc_stim;
  WHILE cec_sync_p0 = D0
  BEGIN
    wait_cn (xstim [chip, chan]);
    cec_select (chip, chan, sync)
  END;
  rv_read := data [oc]
END;

PROCEDURE rv_write (VAR item : DATA);
OWN ic : 0..1;
BEGIN
  cec_select (chip, chan, sync);
  data [ic] := item;
  ic := cec_inc_stim;
  WHILE cec-sync_p2 = 0 D0
  BEGIN
    wait_cn (xstim [chip,chan]);
```

```
        cec_select (chip, chan, sync)
    END
END;
```

It can now be seen that the counter logic on each side of the CEC can be used to support ROUTEs in either direction (but not both), and that each ROUTE can be programmed as either a bb_route or a rv_route. The software build would determine which of these options is chosen.

We have seen how the executive chips and lower level software can be used to execute real time networks. We will now briefly examine the way in which designs may be created in a form suitable for loading into such an execution environment.

The described and illustrated DIA system is preferably used in conjunction with a software/digital system development which will be referred to herein by the acronym DORIS (Data Orientated Requirements Implementation Scheme). The emphasis in DORIS is on the data passed between functions and components in a system. Exchange of data is a unifying theme and, by applying this principle right through from Requirements Analysis to Implementation Execution, traceability throughout the development process is ensured. A very important further advantage is the ability to analyse a proposed implementation for its performance properties. This arises from the distributed nature of the approach which immediately reduces the reliance on dynamically managed shared resources, a well known hazard and one which it may not be possible to resolve satisfactorily in systems where many disjoint processing functions are crammed into a small number of powerful and complex computers, and where communications are multiplexed onto a small number of high bandwidth links.

The essence of the DORIS approach is illustrated in Figure 3. Requirements Analysis leading to top level System Definition is carried out using CORE (COntrolled Requirements Expression), a method which places great emphasis on identifying the information exchanged between well defined functions. Information about CORE may be found in "CORE - A method for Controlled Requirements Specification" by G P Mullery in the Proceedings of Fourth International Conference on Software Engineering, 1979, pp 126-135. The Design phase is carried out in terms of an adapted form of MASCOT. The principal extension to MASCOT is the introduction of type parameters for templates (a template is a design description used to instantiate component elements in a system). The primary motivation for this extension is to allow generic designs for ROUTEs instead of having to create a new ROUTE template for every type of data communicated in this way. The Implementation phase is based on DIA as described herein.

Figure 3 includes two further blocks. Prototyping, Modelling, Simulation and Animations will assist with the creative process of Definition and Design, and with the investigation of proposed solutions by experimental Implementation. Analysis, Verification, Validation and Testing are the means by which the product of a phase of development may be assessed for conformance with previous phases.

The real time network is a form of design abstraction which in principle is free from implementation concerns. In practice, in the field of real time embedded multi processor systems, the design is likely to be quite heavily influenced by performance considerations, and by the way in which the design will have to be mapped into available execution resources. Nevertheless we strive to maximise

abstraction for the clarity, flexibility, generality, maintainability, reusability, etc which this brings.

A DORIS design is expressed as a pure network, with no explicit relationship to execution hardware. The execution environment is separately described using a Hardware Description Language which allows the available processors and memory, and their interconnections, to be defined. A Mapping Description is then used to relate the network to the hardware. This approach offers considerable promise for the development of effective performance analysis tools.

The DORIS design mapping rules are very straightforward:-

a. An ACTIVITY must be contained with a single processor.

b. An IDA design must exist for any inter ACTIVITY communication implied by the mapping of the ACTIVITIEs into the hardware.

The second rules arises because mapping is expressed purely in terms of location of ACTIVITIEs, with the required form of the IDAs, in terms of their distribution in the hardware, being derived from this. For example if a writer communicates with a reader through a ROUTE the IDA design needed depends on whether the two processes are in the same processor, or are in adjacent processors, or are even further apart. The DORIS toolset handles this situation by a Template Substitution technique. In network terms the external specification and the function from the application viewpoint remain the same whatever template is substituted; however the internal design differs and some additional network connections to executive facilities may be needed.

It has been stated that the process interaction properties of a ROUTE remain the same regardless of how the ROUTE is mapped

into the execution hardware. This means that the asynchronous forms remain asynchronous with the same sort of timing constraints or lack of them, and likewise the synchronous forms remain the same in terms of providing buffering or rendezvous characteristics. However information propagation delays will increase as the ROUTE becomes distributed over wider areas. It is only the general nature of the interaction which remains constant, but this is important because it opens the door to generalised timing analysers which can work in terms of timing parameters determined in a direct manner by consideration of the execution environment and network distribution.

Each KEC 46 and each CEC 48 may comprise a custom designed Integrated Circuit. To provide processor independence the chips CEC and KEC may have TTL compatible Inputs and Outputs and be accessed via conventional memory read and write operations.

The KEC supports the co-operative scheduling of activities, within a single processor, under software control and handles the asynchronous external stimuli that may be used to replace the pre-emptive interrupts used in conventional microprocessors. The KEC contains an activity matrix and ripple search logic to identify the next schedulable, primed activity against a fixed rule set. Activities are designated schedulable and primed under software control. Asynchronous external stimuli are latched on chip, but can only be prime activities in the matrix under software control. Not until they have safely primed an activity in the matrix are these latches cleared. The KEC uses an unconventional read strobe to allow the chip logic to operate in parallel, asynchronously, one step ahead of the processor. The KEC uses a novel design in the

"round robin" member search logic within the prioritised set search when selecting the next activity.

As noted, each KEC 47 supports the scheduling of processing tasks, termed activities, for an individual, multi-tasking processor. It will provide, on request, the number of the next activity to be allocated processing time, under executive software control in the presence of external asynchronous stimuli.

By way of example, each KEC 47 might contain support for sixty-four activities, eight of which are associated with external stimuli. Activity numbers can be programmed to be included or excluded as candidates for scheduling, but the next activity number selection rules are fixed. There are eight priority levels with eight activity numbers of each priority. Search logic identifies the next included activity number on a round robin basis, in the highest priority level containing an included activity number.

The CEC enables asynchronous hardware coupling between processor pairs, where each processor may be operating in an independent time frame. Each CEC holds and manipulates variables under software control from each processor and generates an asynchronous external stimulus to each side. Used in conjunction with Asynchronous Dual Port Memory (ADPM), each CEC can support many parallel asynchronous or synchronous software communication routes, established in the ADPM between the processor pairs.

Each CEC 48 may comprise a custom VLSI chip which contains variables and logic to support the concurrent use of various types of shared memory communication mechanisms, between a pair of asynchronously operating processors. The mechanisms steer writing and reading processes to data areas, termed slots, located in the

Asynchronous Dual Port Memory (ADPM) 46, that is connected in parallel with the CEC. For example, the CEC may be designed to support the concurrent use of sixteen inter-processor channels, each channel supporting the concurrent use of: two four-slot mechanisms (one in each direction); two two-slot mechanisms (one in each direction); and a message passing mechanism (that can be used in either direction). Handling for sixty-four stimulii (thirty-two in each direction) may be provided.

Preferably each CEC has two completely independent processor interfaces designated L and R (Left and Right), allowing connection between two asynchronous operating processors, with no mutual access restrictions.

The CEC can be connected between two processors that have independent clocks. Each processor is allowed free access to its side of the CEC, without the need for hardware exclusion, arbitration or synchronisation. The CEC is structured in two halves, with each half containing the circuitry associated with each processor. This consists of stimulus latches, shared bit variables and logic. Each stimulus latch can only be set from one side and can only be copied when identified to the processor. The latches that hold shared bit variables can only be set and cleared from one side, but accessed by the logic from both sides. The logic allows the CEC to search the copied stimulus latches, manipulate the stored variables in a particular fashion under software control and generate the asynchronous external stimulus.

CLAIMS

1. A distributed computer system comprising:-

a plurality of asynchronous computer units each with a data processor and a private data memory linked to the processor by way of a private data bus;

shared data memory means having access ports linked to respective ones of the computer units via the associated private data busses for each computer unit to be able to communicate with any other computer unit by way of a respective two-way data route comprising a respective individual area of shared data memory; and

communication control means connected to the private data busses and the shared data memory means for responding to control data issued by the processor of any computer unit to initiate communication via a route determined by the processor.

2. A system according to claim 1, wherein said shared data memory means comprises a plurality of asynchronously operable dual port memories, one for each said data route, and said communication control means comprises a plurality of communication control units connected to control respective ones of the dual port memories.

3. A system according to claim 1, wherein each of said computer units is programmed with activity scheduling software for initiating periodic reference by the associated processor to a pre-designated memory area and for executing activities selected in dependence upon a data signal located at said pre-designated memory area; the system including activity control means operable for receiving demand signals identifying respective activities to be carried out by respective computer units and for registering said signals in the said pre-designated memory area of the appropriate computer unit.

4. A system according to claim 3, wherein the activity control

means is connected to the communication control means for receiving demand signals identifying activities including responses by each computer unit, to communication requests from other computer units.

5. A system according to claim 3, wherein the activity control means comprises, for each computer unit, a respective individual register unit connected to the computer unit via the associated private data bus and occupying said pre-designated memory area for the computer unit.

6. A system according to claim 1, wherein the shared data memory means and the communication control means are together operable for making available to any two computer units a fully asynchronous form of inter-communication in which each of the two computer units can access the respective shared data memory area out of any form of synchronism with accesses by the other of the two computer units.

7. A system according to claim 6, wherein the computer units are programmed with activity scheduling software and with respective sets of software modules for causing the computer units to execute associated predetermined activities, the modules being selected for running by the scheduling software and the modules including communication processes which are operable for passing data, both between modules which run in the same computer unit and between modules which run in respective ones of two different computer units, by way of a two-way data route comprising a respective area of shared data memory.

8. A system according to claim 6, wherein the shared data memory means and the communication control means are together operable for also making available to any two computer units a less

than fully asynchronous form of inter-communication in which there is a least some synchronisation of accesses to the shared data memory area by the two computer units, which of the fully asynchronous and not fully asynchronous inter-communication forms is made available being determined by the data issued to the communication control means by the computer unit which initiates the communication.

Published 1991 at The Patent Office, Concept House, Cardiff Road, Newport, Gwent NP9 1RH. Further copies may be obtained from Sales Branch, Unit 6, Nine Mile Point, Cwmfelinfach, Cross Keys, Newport, NP1 7HZ. Printed by Multiplex techniques ltd, St Mary Cray, Kent.